Université de Montréal
École des Hautes Études Commerciales
École Polytechnique

Université de Montréal
C.P. 6128, Succursale Centre-ville,
Montréal (Québec) Canada H3C 3J7

Téléphone +1-514-343-7575
Télécopieur (Fax) : +1-514-343-7121
crt@crt.umontreal.ca
www.crt.umontreal.ca/CRT/

CRT-2001-07

Centre de recherche sur les transports (C.R.T.)
Centre for Research on Transportation

A HYBRID GENETIC ALGORITHM FOR THE GENERALIZED TRAVELING SALESMAN PROBLEM

by

Jean-Yves Potvin
Daniel Deladurantaye

Février 2001

102-03122/

PUBLICATION
CRT-2001-07

*A HYBRID GENETIC ALGORITHM FOR THE GENERALIZED TRAVELING SALESMAN PROBLEM**

by

Jean-Yves Potvin[1]
Daniel Deladurantaye[1]

* This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and by the Fonds pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR).

[1] Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7

Dépôt légal – Bibliothèque nationale du Québec
Bibliothèque nationale du Canada, 2001

Résumé. Le problème du voyageur de commerce généralisé consiste à déterminer un cycle hamiltonien de longueur minimale passant par plusieurs sous-ensembles de sommets. Un algorithme génétique hybride est développé pour une variante de ce problème où exactement un sommet doit être visité dans chaque sous-ensemble. Dans cet algorithme, la portion évolutionniste procède à la sélection des sommets, tandis que des techniques classiques de recherche opérationnelle sont utilisées afin de générer une solution à partir des sommets choisis. Des résultats numériques sont présentés sur un ensemble de problèmes tests et une comparaison est établie avec deux heuristiques récentes proposées dans la littérature.

Mots clés: problème du voyageur de commerce généralisé, algorithme génétique.

Abstract. The Generalized Traveling Salesman Problem consists of determining a shortest Hamiltonien cycle passing through each of several clusters of vertices. A hybrid genetic algorithm (GA) is developed to solve a variant of this problem where exactly one vertex must be visited in each cluster. In this algorithm, the GA searches for a good selection of vertices, while classical operations research techniques are used to produce a tour with the selected vertices. Numerical results are reported on a standard set of benchmark problems and a comparison is provided with two recent heuristics reported in the literature.

Keywords: generalized traveling salesman problem, genetic algorithm.

A Hybrid Genetic Algorithm for the Generalized Traveling Salesman Problem

Jean-Yves Potvin and **Daniel Deladurantaye**
Centre de recherche sur les transports
Université de Montréal,
C.P. 6128, succursale Centre-ville,
Montréal, Québec, Canada H3C 3J7

Abstract

The Generalized Traveling Salesman Problem consists of determining a shortest tour on a graph passing through each of several clusters of vertices. A hybrid genetic algorithm (GA) is developed to solve a variant of this problem where exactly one vertex must be visited in each cluster. In this algorithm, the GA searches for a good selection of vertices, while classical operations research techniques are used to produce a tour with the selected vertices. Numerical results are reported on a standard set of benchmark problems and a comparison is provided with the two best heuristics reported in the literature.

1 Introduction

The symmetric Traveling Salesman Problem (TSP) is a canonical NP-hard problem in combinatorial optimization [8]. Given a complete undirected graph with a length associated with each edge, the objective is to find a shortest tour passing through each vertex exactly once. Such a tour is also known as a Hamiltonian cycle. The Generalized TSP (GTSP) extends the classical problem by partitioning the set of vertices into a number of subsets or clusters. Then, a shortest tour passing through at least one vertex in each cluster must be found. Note that when the number of subsets is equal to the number of vertices (i.e., when there is only one vertex in each cluster), the GTSP reduces to the TSP. Applications of the GTSP are reported in many areas such as location-routing, computer design, loop material flow system design and postal box collection [7].

Different variants of the GTSP are found in the literature. Here, we consider problems defined on the Euclidean plane, and where exactly one vertex must be visited in each cluster. More formally let $G = (V, E)$ be an undirected graph where $V = \{1, \ldots, n\}$ is the vertex set, $E = \{(i, j) : i, j \in V, i < j\}$ is the edge set, and a non negative length or distance d_{ij} is associated with every edge (i, j). If the set V is partitioned into m clusters $V_1, \ldots, V_m$, the problem is to find a shortest cycle which contains exactly one vertex in each cluster. Exact algorithms to solve this problem, using branch-and-cut or dynamic programming, are found in [4, 6, 15, 17]. Due to their exponential nature, however, these approaches are restricted to relatively small-sized instances. Recent heuristic approaches are reported in [1, 13, 16].

In the following, Section 2 first presents issues related to solution representation and fitness evaluation for a genetic algorithm (GA). Section 3 then describes the hybrid GA proposed for solving the GTSP problem. Finally, Section 4 reports computational results obtained on a set of benchmark problems. The conclusion follows.

2 Solution representation

Genetic algorithms have been widely used for solving different combinatorial optimization problems, including the TSP [11]. But, to the best of our knowledge, this is the first application of a GA to the GTSP. The procedure that we propose hybridizes the GA with classical operations research techniques. Basically, the GA searches for a good selection of vertices, one in each cluster, while classical operations research techniques are used to produce a solution with the selected vertices (i.e., a good ordering of these vertices on the tour).

When applying a GA to a combinatorial optimization problem, an appropriate represention must first be chosen, given that classical bit-strings are often inap-

propriate. In the following, the chosen representation is introduced and the way to decode it into a solution of the GTSP is explained.

2.1 Encoding

Here, a chromosome is a string of m integers, where the integer in position i corresponds to the vertex selected in cluster V_i, $i = 1, ..., m$. Thus, each chromosome simply represents a set of m vertices, and their ordering is irrelevant at this stage. The true ordering is done subsequently, using operations research techniques (see Section 2.2).

2.2 Decoding

The chromosome is decoded into a solution of the GTSP by ordering its vertices and its fitness corresponds to the value of the solution produced (i.e., the tour length). To order the set of vertices, the farthest insertion heuristic is first applied. Then, local search heuristics based on edge exchanges are performed for further improvement. These methods are briefly described in the following.

Farthest insertion heuristic. The Farthest Insertion (FI) heuristic simply inserts the vertices one by one in the current solution. The next vertex to be inserted is the one which maximizes the minimum distance to the nodes in the current tour. Then, the insertion place is chosen to minimize the detour. More precisely:

1. Select a vertex i at random among the m vertices.
2. Select vertex k which is the farthest from vertex i and form the subtour $i - k - i$.
3. Select vertex k not in the subtour which is the farthest from the nodes in the subtour.
4. Find edge (i, j) in the subtour such that the detour $d_{ik} + d_{kj} - d_{ij}$ is minimal. Insert k between i and j.
5. If all vertices have been inserted, STOP. Otherwise, go to step 3.

It is worth noting that the complexity of FI is $O(m^2)$.

Edge exchanges. The 2-opt [9], Or-opt [10] and 4-opt* [14] local search heuristics are applied one by one, in this order, to the current solution for further improvement. In each case, the search framework is the following:

1. Start with the tour s produced by FI (in the case of 2-opt) or the previous local search heuristic (in the case of Or-opt and 4-opt*).
2. Generate the neighbors of s and select the best one s'.
3. If s' is worse than s, then STOP with s. Otherwise $s \leftarrow s'$ and go back step 2.

The neighborhood of s in step 2 depends on the edge exchange heuristic. In the case of *2-opt*, new solutions are produced by replacing two edges in the current solution by two new edges. This neighborhood is of complexity $O(m^2)$ as it corresponds to the number of ways to select two edges to be removed among m edges. The *Or-opt* considers a subset of 3-opt exchanges, where three edges are replaced by three new ones. Basically, a string of one, two or three consecutive vertices is removed from the current tour and reinserted at another place in the tour. Although the 3-opt neighborhood is of complexity $O(m^3)$, Or-opt is of complexity $O(m^2)$ because it considers only a restricted subset of 3-opt exchanges. Finally, the *4-opt** neighborhood corresponds to a subset of 4-opt exchanges, where four edges are replaced by four new ones. Although the complexity of the 4-opt neighborhood is $O(m^4)$, 4-opt* is of complexity $O(m^2)$ because stringent conditions must be satisfied for an exchange to be valid. In particular, an exchange should not lead to the displacement of a string of customers with a length exceeding a given threshold. This heuristic is rather complicated and the interested reader is referred to [14] for details.

In the next section, the algorithmic framework of the GA is presented and each of its components is described in turn.

3 The algorithm

The algorithmic framework is quite straightforward and can be summarized as follows:

1. Generate an initial population of p solutions.
2. For I generations do:
   - 2.1 Selection;
   - 2.2 Crossover;
   - 2.3 Mutation.
3. Output the best solution found.

3.1 Initial population

The initial population is randomly generated. That is, each new chromosome is created by randomly selecting one vertex in each cluster.

3.2 Selection

Before selecting the parents, a rank-based method is first used to associate a value with each chromosome [3, 19]. The chromosome with highest fitness gets rank 1 and is assigned some predefined MAX value; the chromosome with lowest fitness gets rank p and is assigned some predefined MIN value. In general, the chromosome of rank i is assigned a value v_i between MIN and MAX based on the following formula

$$v_i = MAX - \frac{(MAX - MIN)(i-1)}{p-1} \qquad (1)$$

In our experiments MAX is set to 1.5 and MIN to 0.5, so that the summation over all chromosome values corresponds to the size of the population. Consequently, the selection probability of a chromosome of rank i is

$$p_i = \frac{v_i}{\sum_{i=1}^{p} v_i} = \frac{v_i}{p} \qquad (2)$$

and the expectancy E_i over p selection trials is simply equal to the value v_i, namely:

$$E_i = p\frac{v_i}{p} = v_i. \qquad (3)$$

Note that this approach does not put any emphasis on the magnitude of the fitness gap between two chromosomes; only their relative order is important. In particular, no special emphasis is put on a dominant chromosome with a very high fitness, thus alleviating premature convergence of the population. Once the selection probabilities have been determined through this rank-based method, the parents are selected using Stochastic Universal Sampling (SUS) [2]. As opposed to the classical roulette-wheel selection, this approach provides a lower and an upper bound of $\lfloor v_i \rfloor$ and $\lceil v_i \rceil$, respectively, on the number of selections for the chromosome of rank i.

Once the parents have been selected, they are then processed by the crossover and mutation operators to produce offspring which encode a new choice of vertices.

3.3 Crossover

Here, two parent chromosomes are chosen at random and mated to produce an offspring. This is repeated until p offspring are produced. Given that chromosomes represent sets of vertices, where the ith position encodes the vertex selected in cluster V_i, classical crossover operators can be easily applied. In our experiments, uniform crossover was used [18] where, at each position (cluster), a parent is randomly selected to provide its vertex to the offspring.

3.4 Mutation

In classical GAs, mutation is often considered as a secondary operator aimed at slightly perturbing the search. In our application, however, mutation was found to be a fundamental operator. The simple random mutation schemes that we first developed, in the spirit of classical GAs, never led to implementations that were even close to the best GTSP heuristics. Only the more sophisticated mechanism presented below allowed us to produce competitive results.

The basic mutation mechanism, called mutation *M1*, processes the chromosome position by position (cluster by cluster) and randomly replaces the selected vertex by another one in the same cluster. This procedure has been integrated within a local search scheme to produce mutation *M2* as follows:

1. Set chromosome c as the initial chromosome.
2. While there is an improvement do:

   For $i = 1, .., m$ do:

   2.1 Replace the vertex at position i in chromosome c by a randomly chosen vertex in cluster V_i to produce chromosome c'.

   2.2 Evaluate the impact of this replacement on solution quality.

   2.3 If the solution associated with chromosome c' is better than the one associated with chromosome c, then $c \leftarrow c'$.
3. Output chromosome c.

In Step 2.2, the impact of the move on solution quality is evaluated as follows. In the tour associated with chromosome c, the vertex in cluster V_i is directly replaced by the randomly chosen one to produce a new tour. We then reoptimize it with the 2-opt, Or-opt and 4-opt* edge exchange heuristics. Note that the tour associated with chromosome c is already locally optimal with regard to these exchange heuristics. Consequently, only a few iterations are needed to reach a new local optimum after the replacement of a single vertex. This is much less expensive than recomputing a new solution "from scratch", by applying the FI heuristic to construct an initial tour and then by reoptimizing this initial tour with 2-opt, Or-opt and 4-opt*.

4 Computational Results

To test our algorithm, we used the 36 benchmark problems of Fischetti et al. [4]. These problems are derived from the TSPs found in the TSPLIB library [12] by applying a clustering procedure which partitions the set of vertices into $m = \lceil \frac{n}{5} \rceil$ clusters. The code was written in C++ and the tests were run on a PC equipped with a Pentium II processor (300 MHz).

The results of our GA are compared with those obtained with the GI^3 heuristic [13] and the tabu search heuristic in [16], which are the best heuristics known to date to solve the GTSP. GI^3 is an insertion heuristic, followed by a local reoptimization procedure based on 2-opt and 3-opt edge exchanges. In this algorithm, the insertion and edge exchange moves are "generalized" to consider different choices of vertices. The algorithm in [16] is more powerful as it uses the mechanisms at the core of the tabu search heuristic [5] to escape from local optima. The neighborhood structure is based on the addition and removal of vertices, and allows the exploration of the infeasible domain through penalties in the objective (i.e., solutions with no vertex or more than one vertex in a given cluster are considered).

4.1 Parameter sensitivity

We performed a number of preliminary experiments on randomly generated problems with up to 500 vertices to evaluate the sensitivity of the solutions produced to various parameter values. Our observations are summarized below.

Population size and number of generations. The population size and number of generations were set at 50 and 400, respectively. On the largest problems, we observed a fast improvement in the first 50 to 100 generations, and then a slower improvement up to generation 300-400, approximately. Increasing the number of generations further did not lead to any significant improvements.

Mutation rate. A global mutation rate p_M must first be defined. Then, mutations *M1* and *M2* are applied using probabilities p_{M1} and p_{M2} with $p_{M1} + p_{M2} = 1$. With regard to the global mutation rate, the best results were obtained with $p_M = 0.5$. This is much higher than in classical GA implementations, where a very small mutation rate is often suggested. We also found that varying the probabilities p_{M1} and p_{M2} during the search, rather than keeping constant values, was beneficial. Basically, a higher probability should be associated with *M1* at the start of the search, and a lower probability towards the end. Mutation *M2* is very useful to get competitive results, but its probability should be kept low at the start to avoid premature convergence. In the current implementation, p_{M1}= 0.8 and p_{M2}= 0.2 for 90% of the iterations; these values are then switched to p_{M1}=0.2 and p_{M2}=0.8 for the last 10% of the iterations. More gradual adjustments to these values may be beneficial, but we did not find a formula that produced significantly better results.

Crossover rate. The crossover rate was set to the standard value of 0.6. Smaller values degraded the solutions, while larger values did not significantly impact solution quality.

4.2 Comparison on benchmark problems

Table 1 compare the results produced by our GA with the results reported in [13] and [16] for GI^3 and TABU, respectively. The number at the end of a problem identifier indicates the size of the problem (e.g., EIL51 contains 51 vertices). The column *Best* refers to the best of the three runs, while *Avg.* is the average. In the case of *GI*^{3}, the results reported by Renaud and Boctor in [13] correspond to a single run on each problem instance because there is no stochastic element in their implementation. The values shown are the ratio of the heuristic solution on the optimal one. Therefore, a value of 1.0000 indicates that an optimal solution was found. The column *CPU* is the computation time in seconds. Note that TABU was run on a SUN Sparc 5 and *GI*^{3} on a SUN Sparc LX. Consequently, their CPU times should be divided by 2 and 4, respectively, for a fair comparison with our 300 Mhz PC.

Table 1 shows that our GA implementation is competitive, as it produces solutions within 1% of the optimum on average, like the two other methods. The heuristic *GI*^{3} is the fastest, but leads to solutions that are at (almost) 1% above the optimum. TABU can still be considered as the best approach, since it is faster than GA and generates better solutions, on average. GA exhibits a slightly larger variance from one run to another, as compared with TABU, but this characteristic seems to be beneficial. When the best solution over 3 runs is taken, our algorithm finds a larger number of optimal solutions on the test set (i.e., 24 optimal solutions versus 21) while the average solution values of GA and TABU become very close (i.e, .26% above the optimum for GA versus .20% for TABU, a gap of .06% only). Note that these results were not significantly improved by increasing the number of generations in the GA. For example, after 700 generations, the average percent over the optimum for the best of 3 runs was stable at .26%, with one additional optimal solution found on problem EIL101.

Problem	*GA*			*TABU*			*GI*^{3}	
	Best Length	Avg. Length	CPU	Best Length	Avg. Length	CPU	Length	CPU
EIL51	1.0000	1.0000	3	1.0000	1.0000	17	1.0000	1
ST70	1.0000	1.0010	7	1.0000	1.0000	26	1.0000	2
EIL76	1.0000	1.0000	10	1.0000	1.0000	28	1.0000	2
PR76	1.0000	1.0000	10	1.0000	1.0000	27	1.0000	3
RAT99	1.0000	1.0000	19	1.0000	1.0000	65	1.0000	5
KROA100	1.0000	1.0000	21	1.0000	1.0000	42	1.0000	7
KROB100	1.0000	1.0000	20	1.0000	1.0000	45	1.0000	6
KROC100	1.0000	1.0000	19	1.0000	1.0000	39	1.0000	7
KROD100	1.0000	1.0000	21	1.0000	1.0000	39	1.0000	9
KROE100	1.0000	1.0032	20	1.0000	1.0000	39	1.0000	7
RD100	1.0000	1.0002	20	1.0000	1.0032	61	1.0000	7
EIL101	1.0040	1.0053	23	1.0000	1.0013	64	1.0040	5
LIN105	1.0000	1.0000	24	1.0000	1.0000	35	1.0000	14
PR107	1.0000	1.0000	28	1.0000	1.0000	58	1.0000	9
PR124	1.0006	1.0033	41	1.0000	1.0016	81	1.0043	12
BIER127	1.0000	1.0310	55	1.0004	1.0051	56	1.0555	36
PR136	1.0000	1.0091	59	1.0001	1.0024	152	1.0128	13
PR144	1.0008	1.0008	62	1.0000	1.0001	105	1.0000	16
KROA150	1.0004	1.0006	68	1.0000	1.0001	179	1.0000	18
KROB150	1.0000	1.0030	68	1.0000	1.0042	107	1.0000	14
PR152	1.0000	1.0000	96	1.0000	1.0000	85	1.0047	18
U159	1.0000	1.0000	93	1.0000	1.0049	93	1.0260	19
RAT195	1.0012	1.0035	150	1.0012	1.0105	194	1.0000	37
D198	1.0049	1.0064	229	1.0049	1.0062	143	1.0060	60
KROA200	1.0000	1.0000	167	1.0072	1.0073	157	1.0000	30
KROB200	1.0041	1.0160	169	1.0035	1.0059	226	1.0000	36
TS225	1.0000	1.0012	250	1.0009	1.0034	364	1.0061	89
PR226	1.0000	1.0000	247	1.0000	1.0035	142	1.0000	26
GIL262	1.0000	1.0125	407	1.0128	1.0194	319	1.0503	115
PR264	1.0000	1.0022	408	1.0015	1.0034	323	1.0036	64
PR299	1.0015	1.0071	580	1.0035	1.0088	638	1.0223	90
LIN318	1.0000	1.0076	745	1.0010	1.0010	301	1.0459	207
RD400	1.0260	1.0309	1562	1.0105	1.0186	1533	1.0123	404
FL417	1.0112	1.0114	1890	1.0048	1.0048	461	1.0048	427
PR439	1.0129	1.0147	2208	1.0107	1.0148	867	1.0352	611
PCB442	1.0273	1.0508	2285	1.0075	1.0111	1167	1.0591	568
Avg.	1.0026	1.0062	336	1.0020	1.0039	230	1.0098	83
Nb. Optima	24	14		21	13		20	

Table 1: Computational results on the benchmark problems

5 Conclusion

In this paper, a genetic algorithm for solving the GTSP was presented and its competitiveness with the best heuristics known to date was empirically demonstrated on a set of benchmark problems. To obtain such results, the GA had to be hybridized with classical operations research techniques, to produce an ordering (i.e., a tour) with the vertices selected by the GA. A powerful mutation mechanism, based on local search, was also required.

Acknowledgments. Financial support for this work was provided by the Canadian Natural Sciences and Engineering Research Council (NSERC) and by the Quebec Fonds pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR). This support is gratefully acknowledged.

References

[1] Andresol R., M. Gendreau and J.-Y. Potvin, "A Hopfield-Tank Neural Network Model for the Generalized Traveling Salesman Problem", in *Meta-Heuristics - Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, S. Martello, I.H. Osman, C. Roucairol eds., Kluwer, 393–402, 1999.

[2] Baker J., "Reducing Bias and Inefficiency in the Selection Algorithm", in *Proceedings of the Second Int. Conf. on Genetic Algorithms*, Lawrence Erlbaum, 14–21, 1987

[3] Baker J., "Adaptive Selection Methods for Genetic Algorithms", in *Proceedings of an Int. Conf. on Genetic Algorithms and their Applications*, Lawrence Erlbaum, 101–111, 1985.

[4] Fischetti M., J.J.S. Gonzalez and P. Toth, "A Branch-and-Cut Algorithm for the Symmetric Generalized Traveling Salesman Problem", *Operations Research 45*, 378–394, 1997.

[5] Glover F. and M. Laguna, *Tabu Search*, Kluwer, 1997.

[6] Laporte G. and Y. Nobert, "Generalized Traveling Salesman Problem through n Sets of Nodes: An Integer Programming Approach", *INFOR 21*, 61–75, 1983.

[7] Laporte G., A. Asef-Vaziri and C. Sriskandarajah, "Some Applications of the Generalized Traveling Salesman Problem", *Journal of the Operational Research Society 47*, 1461–1467, 1996.

[8] Lawler E.L., J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys eds., *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, 1985.

[9] Lin S., "Computer Solutions of the Traveling Salesman Problem", *Bell System Technical Journal 44*, 2245–2269, 1965.

[10] Or I., "Traveling Salesman-type Combinatorial Problems and their Relation to the Logistics of Blood Banking", Ph.D. Thesis, Dept. of Industrial Enginnering and Management Sciences, Northwestern University, 1976.

[11] Potvin J.-Y., "Genetic Algorithms for the Traveling Salesman Problem", *Annals of Operations Research 63*, 339–370, 1996.

[12] Reinelt G., "TSPLIB - A Traveling Salesman Problem Library", *ORSA Journal on Computing 3*, 376–384, 1991.

[13] Renaud J. and F.F. Boctor, "An Efficient Composite Heuristic for the Symmetric Generalized Traveling Salesman Problem", *European Journal of Operational Research 108*, 571–584, 1998.

[14] Renaud J., F.F. Boctor and G. Laporte, "A Fast Composite Heuristic for the Symmetric Traveling Salesman Problem", *INFORMS Journal on Computing 8*, 134–143, 1996.

[15] Saksena J.P., "Mathematical Model of Scheduling Clients through Welfare Agencies", *CORS Journal 8*, 185–200, 1970.

[16] Semet F. and J. Renaud, "A Tabu Search Algorithm for the Symmetric Generalized Traveling Salesman Problem", Technical report CRT-99-19, Centre de recherche sur les transports, Université de Montréal, 1999.

[17] Srivastava S.S., S. Kumar, R.C. Garg and P. Sen, "Generalized Travelling Salesman Problem through n Sets of Nodes", *CORS Journal 7*, 97–101, 1969.

[18] Syswerda G., "Uniform Crossover in Genetic Algorithms", in *Proceedings of the Third Int. Conf. on Genetic Algorithms* Morgan Kaufmann, 2–9, 1989.

[19] Whitley D., "The GENITOR algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best", in *Proceedings of the Third Int. Conf. on Genetic Algorithms*, Morgan Kaufmann, 116–121, 1989